The Thesis Committee for Sylvia D. Carroll
Certifies that this is the approved version of the following thesis:

3D image processing and FPGA implementation for optical coherence tomography

APPROVED BY
SUPERVISING COMMITTEE:

Supervisor:

Thomas E. Milner

H. Grady Rylander, III

3D image processing and FPGA implementation for optical coherence tomography

by

Sylvia D. Carroll, B.S.E.E., B.S.

Thesis

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

August 2013

Dedication

To my parents, thanks for supporting my academic endeavors; to Dr. Milner, thanks for giving me a chance; and to Dr. Akinwande, for opening the door of opportunity.

Abstract

3D image processing and FPGA implementation for optical coherence tomography

Sylvia D. Carroll, M.S.E.

The University of Texas at Austin, 2013

Supervisor: Thomas E. Milner

This thesis discusses certain aspects of the noninvasive imaging technique known as optical coherence tomography (OCT). Topics include three-dimensional image rendering as well as application of the Fast Fourier Transform to reconstruct the axial scan as a function of depth. Implementations use LabVIEW system design software and a Xilinx Spartan-6 field-programmable gate array (FPGA). The inherent parallel-processing capability of an FPGA opens the possibility of designing a "super-sensor" which entails simultaneous capturing of image and sensor data, giving medical practitioners more data for potentially improved diagnosis. FPGA-based processing would benefit many methods of characterizing biological samples; OCT and photonic crystal microarray biosensors are discussed.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Optical coherence tomography (OCT) is a noninvasive imaging technique that creates images of a sample by detecting backscattered light using a low-power near-infrared light source. This optical analog of ultrasound has garnered widespread interest due to its high resolution which surpasses that of MRI or PET.[1] In addition, other imaging modalities prove impractical for primary care diagnostics. Another advantage of OCT is that it uses small-diameter optical fibers, a characteristic that lends itself to miniaturization within needles, catheters, endoscopes, microscopes and handheld scanners.[2] OCT also features real time data acquisition, but in order to fully utilize that capability the processing speed must be real-time as well. One way to improve processing speed is by performing computations on a field-programmable gate array (FPGA).[3]

The main advantage of FPGA-based OCT is the potential speed increase; advantages of faster OCT systems include the possibility of recording moving objects such as human organs, as well as reducing the delay between data acquisition and display. Another advantage is size or footprint reduction, resulting in portable handheld systems. According to Suzuki (2011),[1] FPGA usage resulted in swept-source OCT (SS-OCT) speeds four times faster than a conventional design. This type of OCT consists of a scanning laser, an analog-digital converter (ADC) for data acquisition, and image processing such as fast Fourier transform (FFT), interpolation, and DC offset. High-speed acquisition and synchronization are necessary. In a conventional system, the host computer handles processing as well as laser tuning, creating a speed bottleneck. For Suzuki's device, moving these processing tasks to the FPGA resulted in an image display rate of 40 frames per second, compared to the previous rate of 10 frames per second. In

addition, the FPGA-based system is half the size of the conventional one.[1] Ustun et al. (2008)[3] also implemented Fourier-domain OCT using an FPGA to perform various calculations such as background subtraction, spectrum interpolation, high-pass filtering, dispersion compensation, and FFT. They achieved a video-quality frame rate of 27 frames/s.

Many current devices would have been even better if processing was FPGA-based. One example is Jung et al. (2011)[2], who explored handheld OCT as an improvement to the otoscope and the ophthalmoscope.

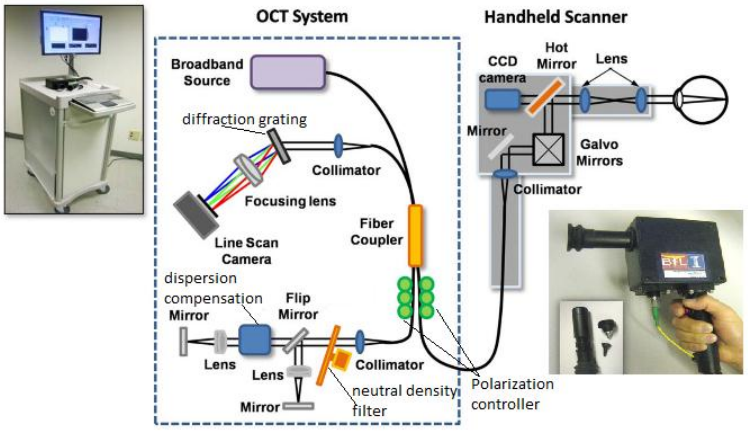


Figure 1: Cart-based SD-OCT system (left) and handheld scanner (right).[2]

Their device (Fig. 1) enables subsurface imaging of the ear (tympanic membrane and ear canal) and eye (anterior chamber and retina) in a primary care setting. Spectral-domain OCT (SD-OCT) was implemented with a four-port Michelson interferometer configuration, and the entire system was contained by a portable medical cart with dimensions of 66 x 50 x 94 cm (length, width, height). The handheld scanner was 11.5 x 11.5 x 6.3 cm. System features include a long optical path for retinal imaging, a short

path for other tissues, and an integrated mini-camcorder for precise positioning. These authors obtained a resolution of ~4 microns (axial) and ~15 microns (transverse), and an image rate of 70 frames/s. According to the authors, this speed is insufficient for clinical 3D *in vivo* imaging.[2]

An FPGA consists of many logic cells, each of which is a small lookup table (LUT), a D-type flipflop and a 2-to-1 mux to bypass the flipflop if needed. The LUT can implement any logic function. Complex logic functions can be produced by connecting many logic cells. FPGAs feature general interconnects as well as fast dedicated lines such as carry chains (Fig. 2) which enable high-speed arithmetic with low logic usage.[4] In addition, modern FPGAs also include dual-ported Block RAMs and embedded DSP blocks which are useful in OCT. The BRAMs are used in resampling, and they also store a LUT containing the logarithmic scaling rather than calculating it at runtime, and the DSP blocks perform multiply-accumulate operations necessary for the Fast Fourier Transform (FFT) algorithm.[5]

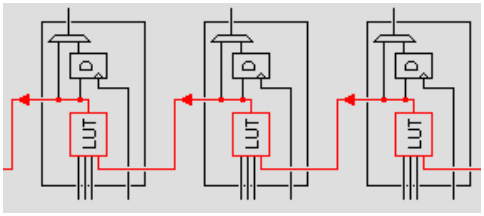


Figure 2: Three FPGA logic cells in a carry chain.[4]

A swept-source FD-OCT configuration (Fig. 3) is well suited to FPGA implementation because less memory is required, as individual lines of data are processed as opposed to frames. Multiple lines can be processed at once because of the fine-grained parallelism and replicated pipelines inherent in FPGA technology. In this type of OCT, a

narrow-band light source sweeps over a broadband spectrum. At each wavelength, interference fringes are produced and a photodetector serially converts the spectral frequency signal into voltage at a rate of roughly 5.2 MHz per line. Then, an analog-to-digital converter (ADC) converts the voltages so they can be processed.[5]

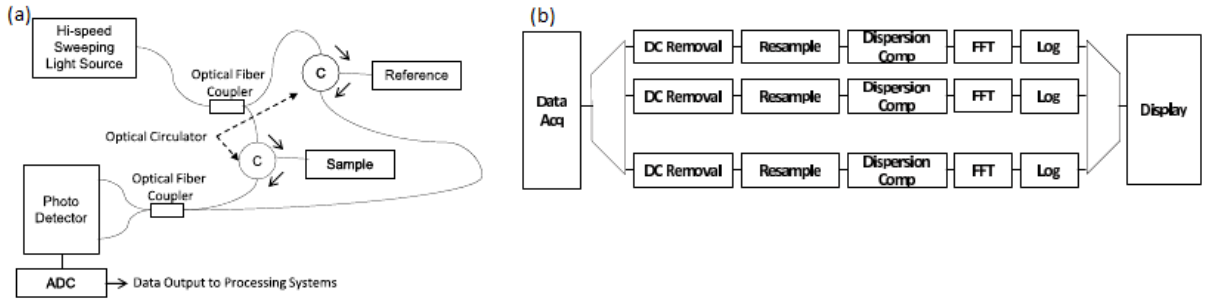


Figure 3: (a) Swept-source OCT; (b) Parallel pipelining in FPGA OCT.[5]

It is also possible to construct three-dimensional images from OCT data, by stacking many 2D images on top of one another, as shown in Fig. 4. Each frame is a cross-sectional (B) scan in the x-y plane. These B-scans are composed of A-scans, which are high-resolution 1d scans in the z direction.[6]

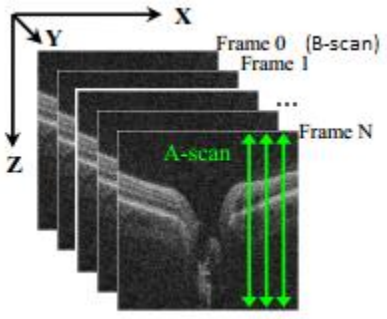


Figure 4: Constructing a 3D image from OCT data. [6]

Chapter 2: Implementation

3D image processing for OCT data

The system was implemented in software using LabVIEW as well as the 3D Viewer feature of ImageJ. A kidney stone ablation dataset (file name: "Jen's data 5-9-2012 HH5 RCA3 not inked time 2.03.30 PM") was used, however the methods are applicable to any dataset. Smaller side tasks included researching methods of LabVIEW instrument control via Ethernet, as well as helping the group implement an OCT system in C++. Working with the 3D data consisted of rendering images as well as improving image quality by filtering. The discussion is chronological, from least recent to most recent.

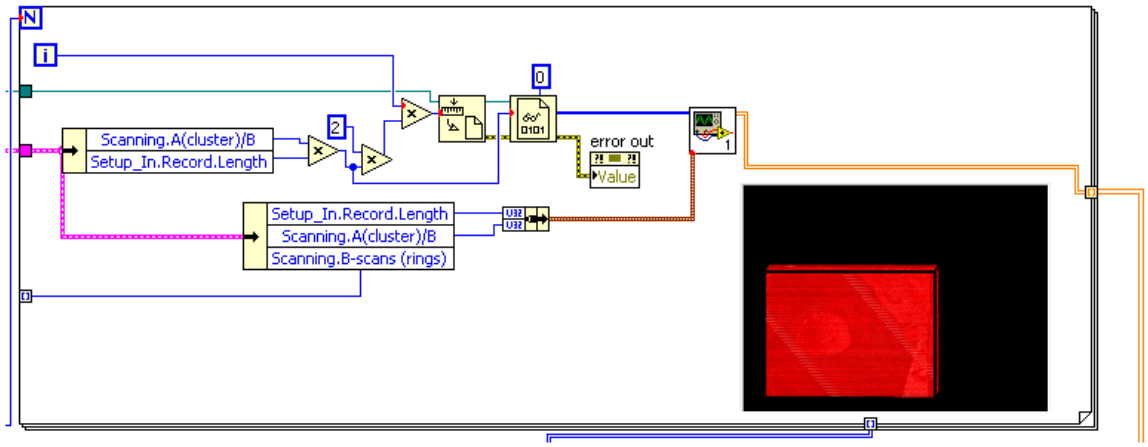


Figure 5: LabVIEW ActiveX 3D rendering of OCT dataset.

A first attempt was to use the ActiveX ComponentWorks 3D rendering available in Labview (CWPlots3D and CWGraph3D) but it runs extremely slow and the resulting image has poor contrast, so ultimately the ImageJ program was used after trying various programs such as ImageVis3D. This will be discussed further on. The 3D binary file reading algorithm is implemented in LabVIEW as shown in Figure 6 and takes place

inside a for loop. The number of loop iterations is the number of images. For each iteration, the binary file is read starting from a position given by 2*(*LoopIndex*)*(*AScansPerB*)*(*NumberOfImages*) and the number of data elements read is given by (*AScansPerB*)*(*NumberOfImages*). For the dataset that was used, *AScansPerB*=1280, and *NumberOfImages*=300. The resulting 1D array is processed by Cooley-Tukey mixed-radix FFT[26], which results in a 2D intensity array which is indexed at the for loop output, giving a 3D array.

We know Figure 6 is really the 3D image because when turned on its side, the en face image is visible as Figure 6 shows. However, there was still doubt, so it was suggested that a binary file depicting a sphere should be made. The sphere was generated, however the programs that are really good at making spheres lacked the memory to handle the OCT dataset (1280x300x300), so this approach did not help much.

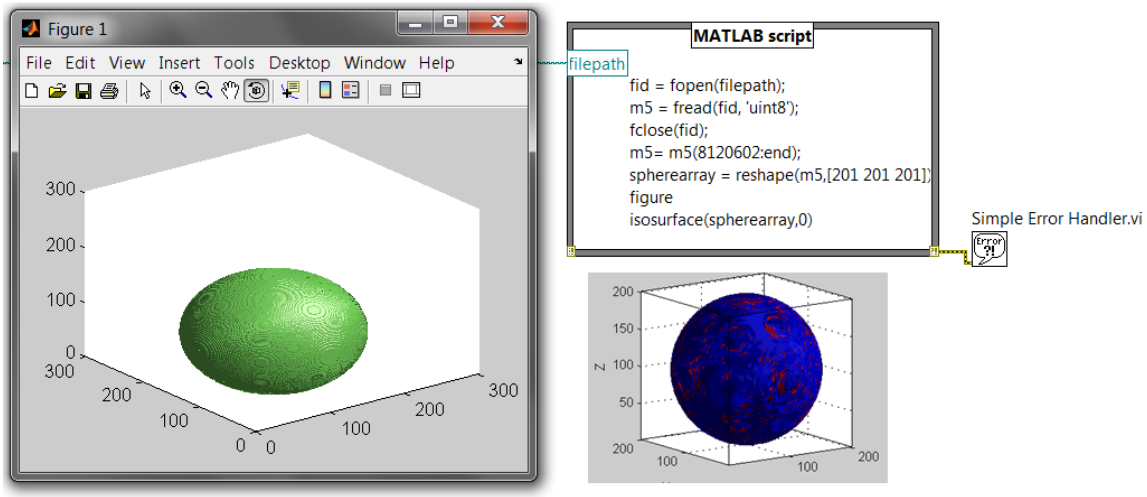


Figure 6: Sphere in MATLAB (left) and Sliceomatic (right).

Even after adding a color map, the poor quality still persisted (see Fig. 7). As a result, alternative programs were sought. One option was ImageVis3D, but the 64 bit version just crashed while the 32-bit version displayed a blank screen.

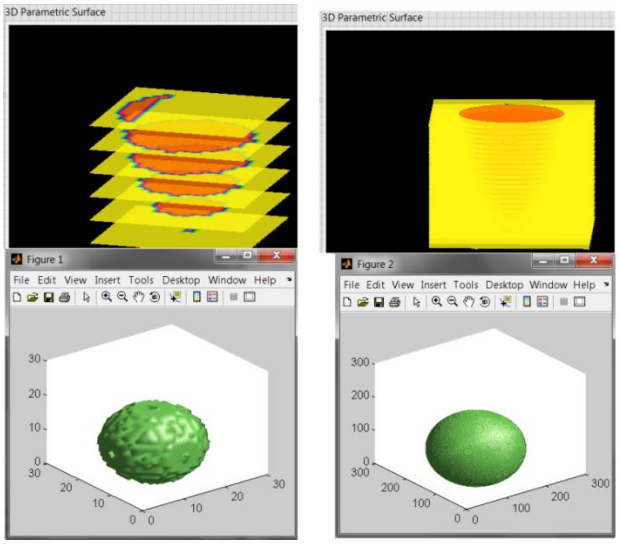


Figure 7: Smaller sphere at left (21x21x21), larger sphere at right (201x201x201) and how each sphere looks in LabVIEW (top) and MATLAB (bottom).

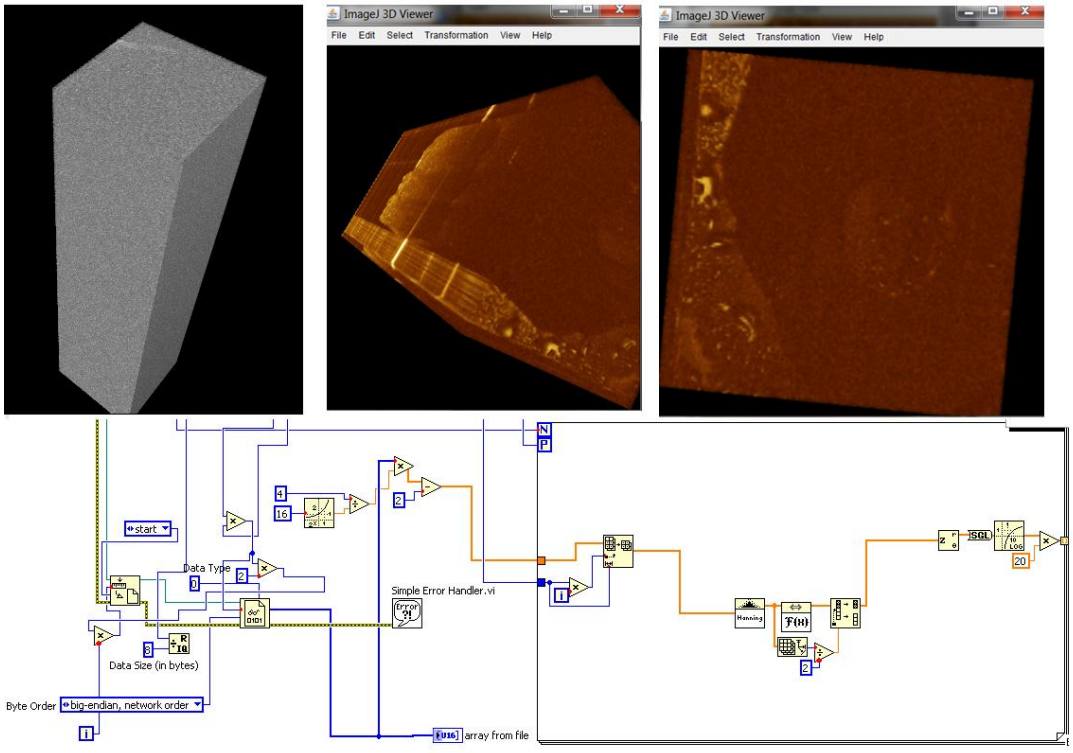


Figure 8: ImageJ renderings of kidney stone ablation. (top left) without FFT, (top middle and right) with FFT, (bottom) FFT algorithm.

The next attempt involved getting ImageJ to display the OCT data but there was no image at first (see Fig. 8); this was because the data needed to be processed by Fast Fourier Transform (FFT) before the image would be visible. A VI was constructed that would perform the conversion, accepting raw binary as input and producing another binary as output, the difference being that the output bin could be read directly by ImageJ. It is also possible to use LabVIEW to start ImageJ, as shown in Fig. 9 which is adapted from a VI from the University of Twente. However, this was not used as it is simple enough for the user to open ImageJ separately. An animated gif image was also created. The Labview FFT algorithm, pictured in Fig. 8, proceeds as follows. First we apply the following operation to the data: $(data)*(4/2^{16})-2$, then take an array subset with

index (*PointsPerAScan*LoopIndex*) and length (*PointsPerAScan*).We then apply a Hanning window, compute FFT, take the first half of the array, get the real portion of the complex number, convert it to single-precision float, take the log base 10 and multiply by 20.

Figure 9: Getting ImageJ to run from LabVIEW.

3D data filtering

The next step was to try to filter the 3d images to make them look better. The first attempt utilized the median filtering VI that is included with LabVIEW, with a filter rank of 7 (see Fig. 10).

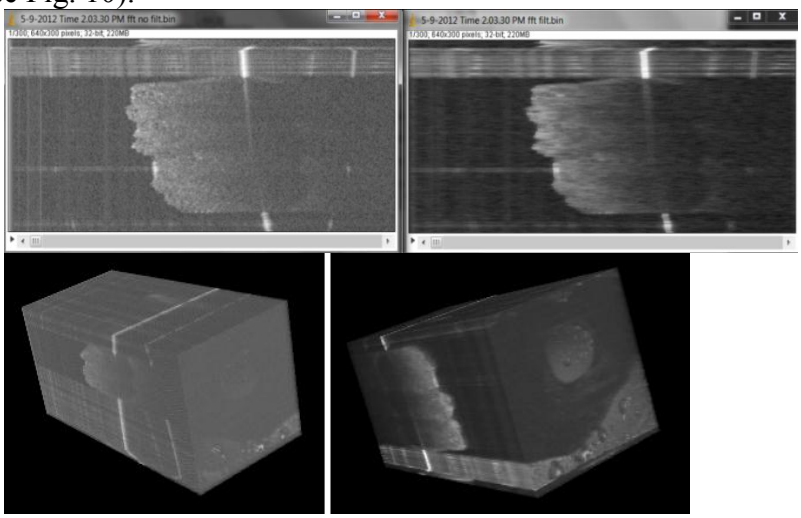


Figure 10: Before and after median filtering (before=left, after=right). Top row is 3D stack, bottom row is a 2D image (60/300).

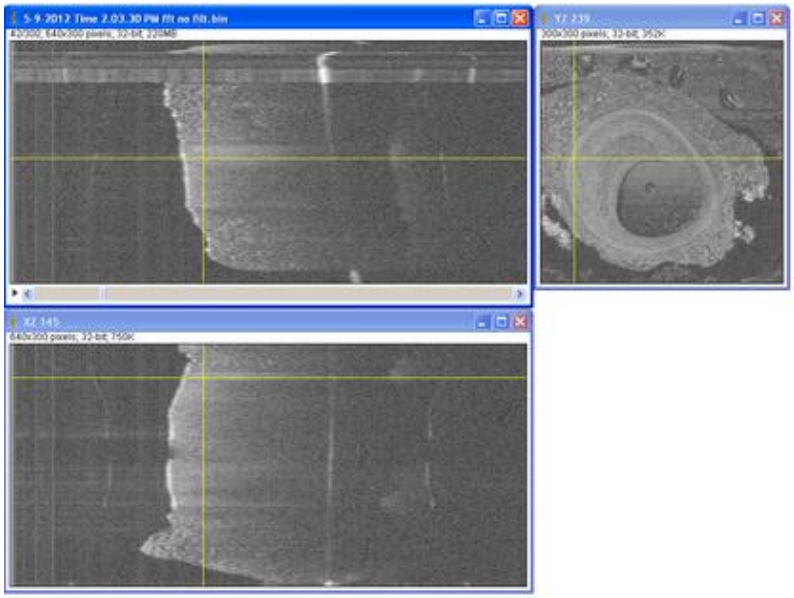


Figure 11: Orthogonal views of 3D rendering of kidney ablation dataset.

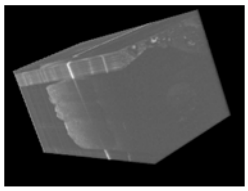
Another feature of ImageJ is Image>Stacks>Orthogonal Views. Different views can be obtained by moving the crosshairs (see Fig. 11). An alternative approach to despeckling is to create a mask or "negative" of the noise and OCT object, then multiply the mask by the OCT image. The idea was to take an NxM window, move it through a B-scan, compute the standard deviation divided by the mean (SD/mean) of the NxM window and plot display that for 3x3, 5x5, 3x5, and 5x3. Regions of the image where tissue is present should exhibit higher levels of standard deviation compared to the regions without tissue. Then we threshold the SD/mean image to create the binary mask, then multiply that by the original image. SD/mean was not the only filter; the same idea could be used with an SD filter, mean filter, and median filter. The Appendix contains the pseudocode for my attempt (3x5 window) followed by the LabVIEW translation. However, this version was not used, instead Austin McElroy's filter implementation was

used to generate Tables 1 and 2. To create filter masks, the 2D input array is fed through two nested for loops. The user can set the x- and y-kernel sizes, and the type of filtering to be done. For each iteration *i*, we take the portion of the array with x and y indices from *(i*-(*kernel size*/2)) to (*kernel size*). Then the 2D array is converted into a 1D array for processing. For the SD filter, we replace all values with the standard deviation of the values. For the mean filter, we replace all values with the mean of the values. Finally, we convert the array back to 2D and output it. For the SD/mean filter, we compute the absolute value of SD/mean. If it is less than or equal to 10, we replace all values with 60, else we replace all values with -60. Finally, for the median filter we replace all values with the median of the values. The filter masks are shown in Tables 1 and 2, as well as the result of multiplying each filter mask by the original unfiltered image.

The code shown in the Appendix uses a 3x5 window. The number of rows (*numrows*) corresponds to *AScansPerB*=300, and the number of columns (*numcols*) corresponds to half the length, that is half of 1280 which is 640. The index variables, *rowindex* and *colindex*, are both initialized to zero. Inside a while loop, we take an array subset with *rowindex* and *colindex* and lengths of 3 and 5. This subset is the window, which we filter and insert back into the image array. We increment *colindex* by 1 every loop iteration, but only increment *rowindex* by 3 if we ran out of columns, and then reset *colindex*. The Appendix contains a diagram of the windowing.

To reiterate, the version shown in the Appendix was not used because Austin's version performs much better (simpler and works without troubleshooting). However, his SD/mean filter required a little tweaking in order to obtain an image; specifically, code was added saying if |SD/mean|<10 make the value -60, else make it +60. It isn't much of a filter though, just increases contrast. Table 1 shows the 3D images, while Table 2 shows

a 2D image, (60/300) from the dataset. If the filter just multiplies by 10 instead of rounding to +60, a faint outline is visible similar to the SD filter but not as good.



(Unfiltered image for comparison)

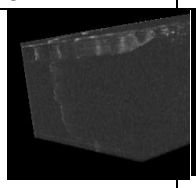
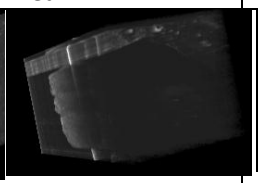
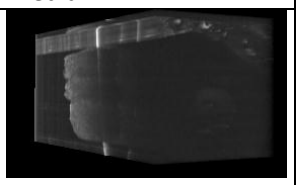
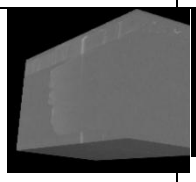
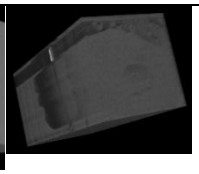
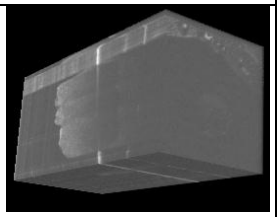
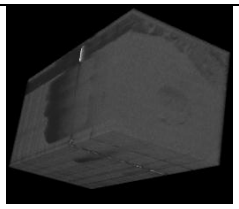
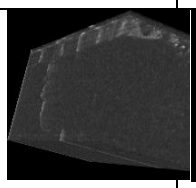
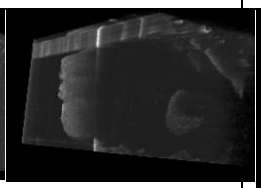
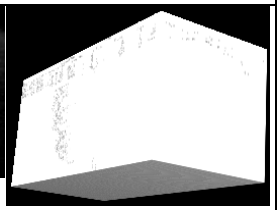
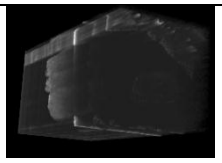
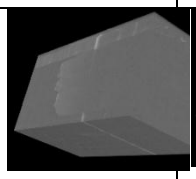
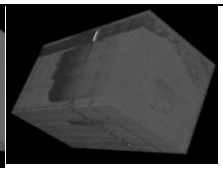
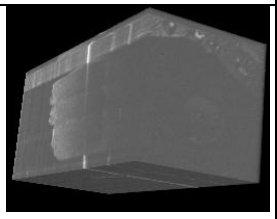
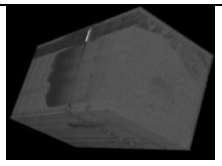
		Filter			
Window		SD	Mean	SD/mean	Median
3x3	Mask only				
	Mask times orig				
3x5	Mask only				
	Mask times orig				

Table 1: 3D image with various filters and window sizes. (continued on next page)

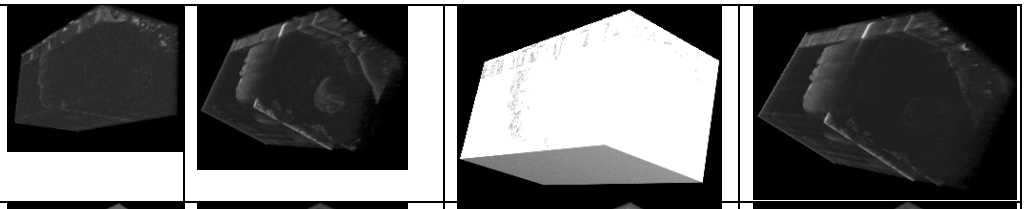
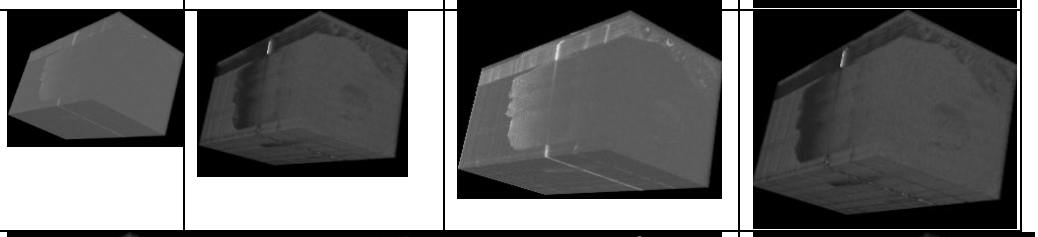
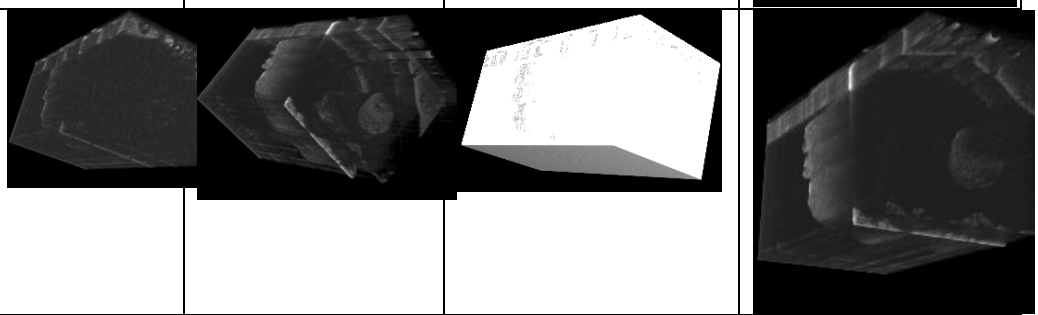
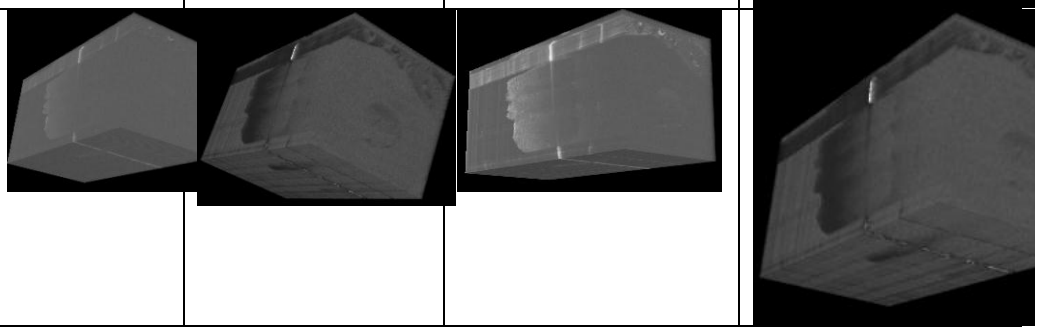
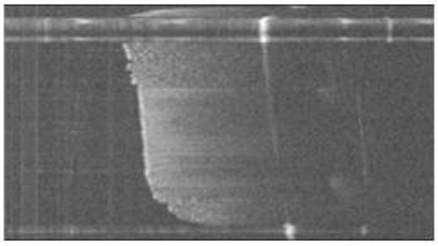
5x3	Mask only	
	Mask times orig	
5x5	Mask only	
	Mask times orig	

Table 1 (continued): 3D image with various filters and window sizes.



Unfiltered image 60 for comparison

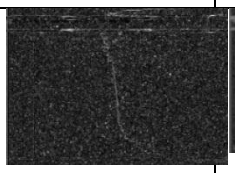
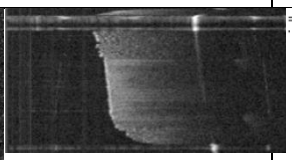
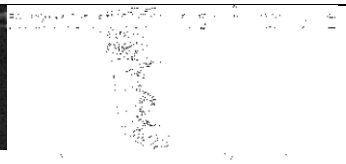
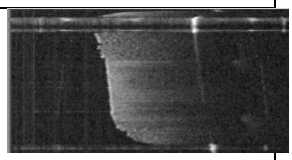
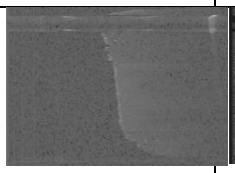
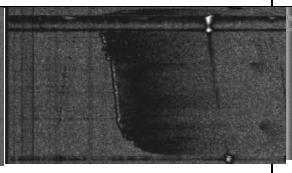
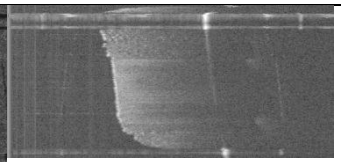
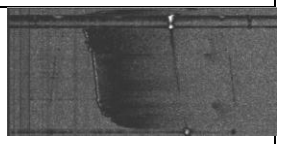
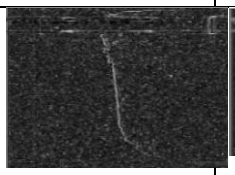
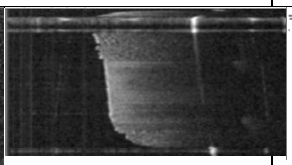
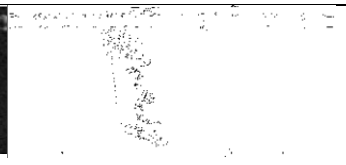
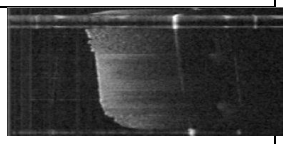
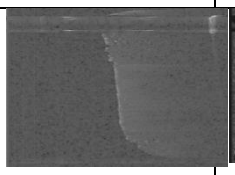
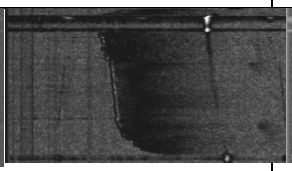
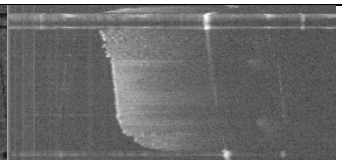
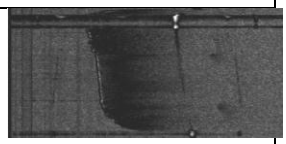
		Filter			
Win-dow		SD	Mean	SD/mean	Median
3x3	Mask only				
	Mask times orig				
3x5	Mask only				
	Mask times orig				

Table 2: 2D image with various filters and kernel sizes. (continued on next page)

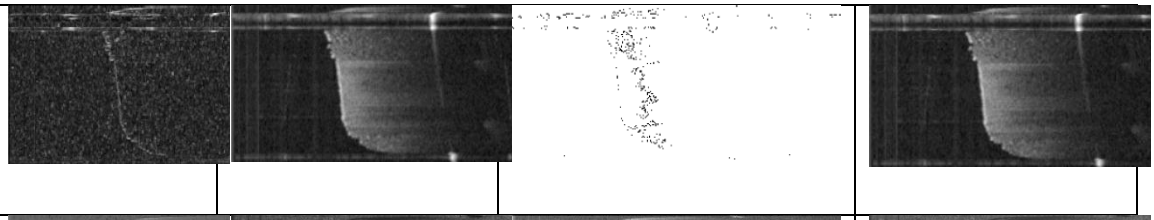
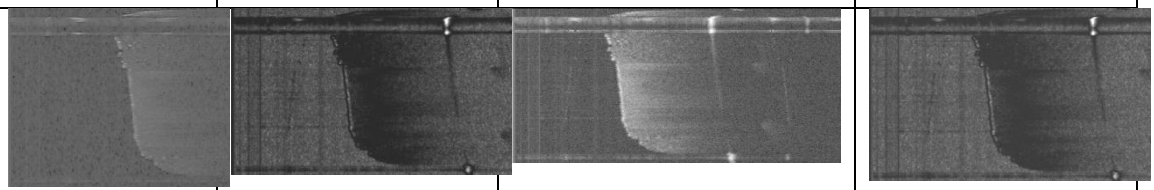
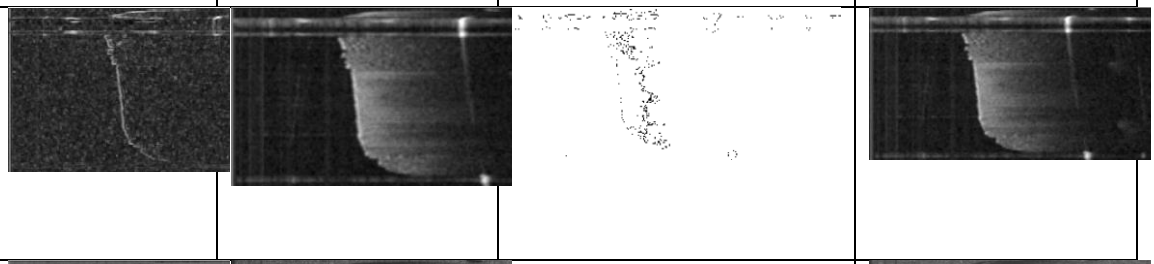
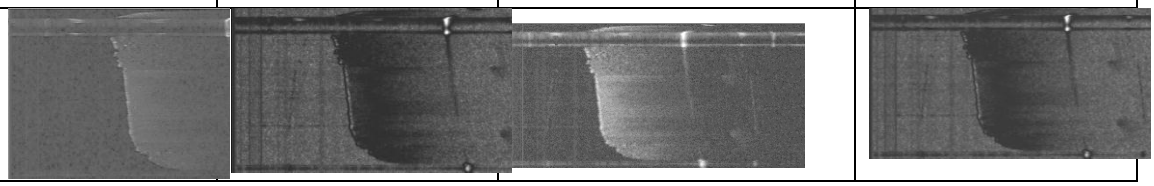
5x3	Mask only	
	Mask times orig	
5x5	Mask only	
	Mask times orig	

Table 2 (continued): 2D image with various filters and kernel sizes.

FPGA implementation of FFT

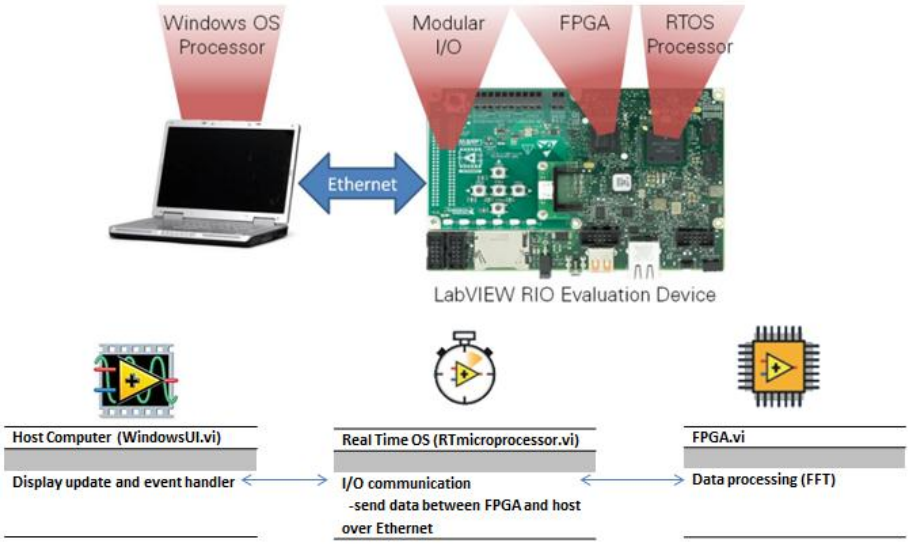


Figure 12: System diagram for NI RIO Evaluation Kit.[7]

The hardware was a National Instruments Reconfigurable I/O (RIO) Evaluation Kit, which had a single-board RIO (NI sbRIO-9636). As shown in Fig. 12, the system had three major components:

- Windows host computer for user interface;
- FPGA for computation;
- Real-time OS for transferring data between host and FPGA.[7]

The FPGA was a Xilinx Spartan-6 LX45. The software was NI LabVIEW 2012 with the FPGA Module, Real Time Module, and FPGA Compile Farm Toolkit. In addition the Xilinx LogiCORE™ IP Fast Fourier Transform was utilized, which implements the Cooley-Tukey FFT algorithm, a computationally efficient method for calculating the Discrete Fourier Transform (DFT).[8]

The LabVIEW FPGA Module also contains a native implementation of FFT. Data transfer is facilitated by Direct Memory Access (DMA), in which the FPGA treats the host RAM as its own. A first-in, first-out (FIFO) architecture is used. This means the data structure holds and releases data in the order they were received.[21] The FFT length, N, must be a power of two. The terms are separated into even and odd N. The math below shows how the N-point Fourier transform sequence splits into two N/2-point Fourier transform sequences. Not only that, but if a term has an index greater than N/2, such as (m+(N/2)), it has the same value as the term with index m, further reducing the computational requirements.[22]

$$X(j) = \sum_{k=0}^{N-1} A(k) \cdot W_N^{jk};\ where\ j = 0, 1, ..., N-1$$
$$A(k)\ complex,\ W_N = e^{2\pi i/N}$$
Separate odd k from even k: $X(j) = \sum_{k\ even} A(k) \cdot W^{jk} + \sum_{k\ odd} A(k) \cdot W^{jk}$
$$\sum_{r=0}^{N/2-1} A(2r) \cdot W_N^{j(2r)} + \sum_{r=0}^{N/2-1} A(2r+1) \cdot W_N^{j(2r+1)}$$
$$\sum_{r=0}^{N/2-1} A(2r) \cdot W_N^{j(2r)} + \sum_{r=0}^{N/2-1} A(2r+1) \cdot W_N^{j(2r)} W_N^j$$
$$\sum_{r=0}^{N/2-1} A(2r) \cdot W_N^{j(2r)} + W_N^j \sum_{r=0}^{N/2-1} A(2r+1) \cdot W_N^{j(2r)}$$

We can keep splitting each series in half recursively until we get two 2-point transforms. The equations are shown below, and Figure 13 shows the graphical representation known as the butterfly.[22]

$$X(0) = W_2^0 A(0) + W_2^0 A(1) = A(0) + W_2^0 A(1)$$
$$X(1) = W_2^0 A(0) + W_2^1 A(1) = A(0) + W_2^1 A(1)$$

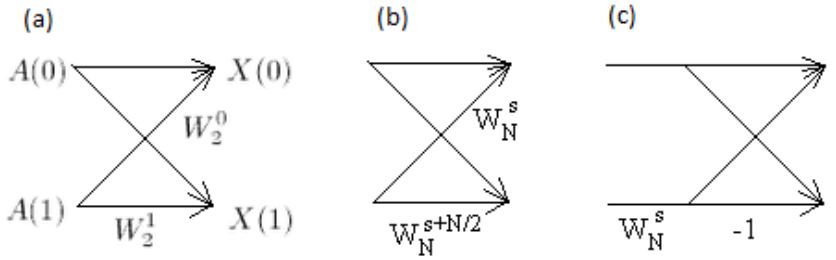


Figure 13: FFT butterfly. (a) Two-point FFT; (b) generalized; (c) simplified.[22]

Figure 13(b) shows the general form of the butterfly, while Figure 13(c) shows the simplification made possible by the following identity:

$$W_N^{s+N/2} = W_N^s W_N^{N/2} = -W_N^s$$
$$since\ W_N^{N/2} = e^{-2\pi i(N/2)/N} = e^{-\pi i} = \cos(-\pi) + i \cdot sin(-\pi) = -1$$

A four-point FFT would require two stages of butterflies. In general, the number of stages equals the exponent m in N = 2m. The outputs of one level are inputs of the next level. Each stage requires N real or N/2 complex multiplications, N/2 sign inversions (multiplication by -1), and N complex additions.[22] LabVIEW FPGA FFT implementation is shown in Figure 14.

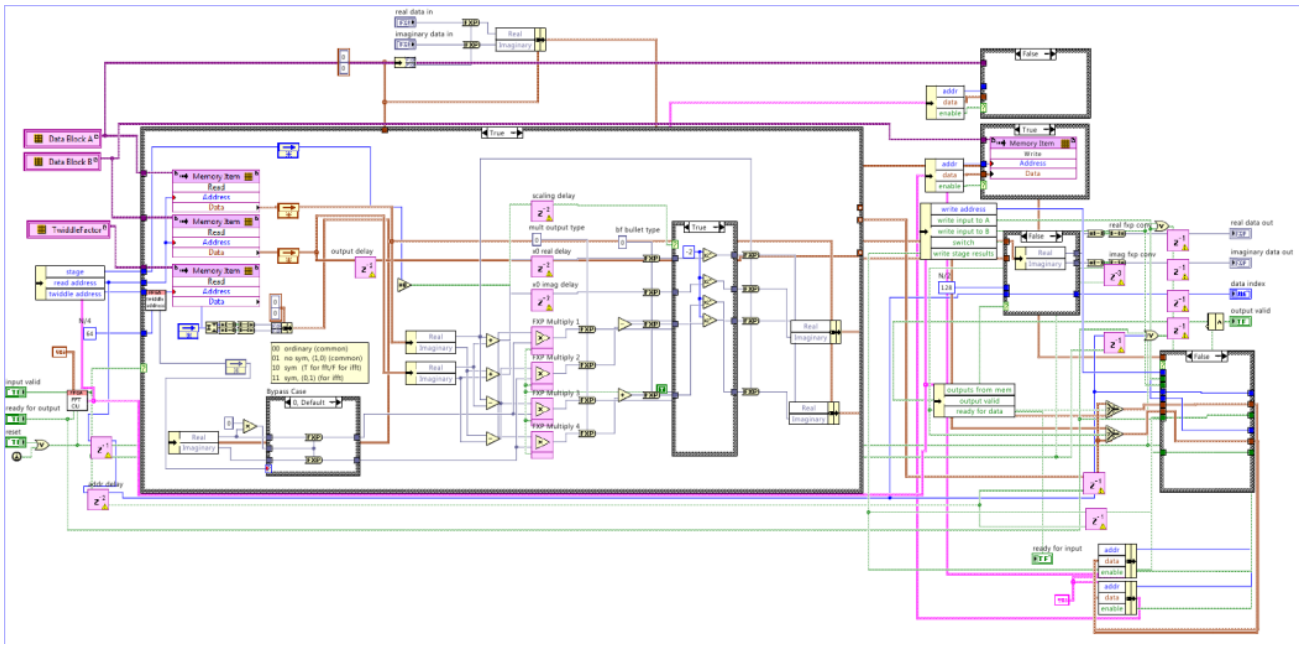


Figure 14: LabVIEW FPGA FFT implementation.

Input and output graphs are shown in Fig. 15. The input graph shows the interferogram of the sample and reference path. A voltage from the photodetector is

measured, but the A/D converter scales the +/- 2 volt range to 0-2^{16}. The output graph shows the data processed by FFT, giving spatial frequency in optical path length domain. In this graph, vertical axis values correspond to path length difference between the sample and reference paths. (J.C. Dwelle and A.B. McElroy, personal communication, April 5, 2013)

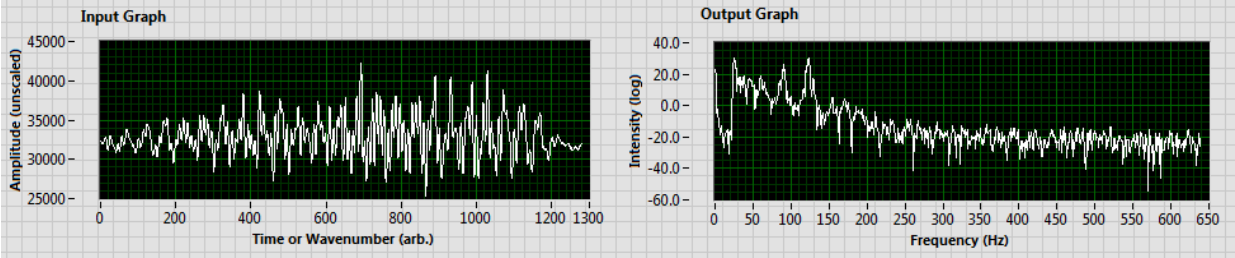


Figure 15: FFT input and output.

National Instruments performed some benchmarking tests on various FPGAs and found the following for the Virtex-5 LX50. A maximum rate of 90 MHz was measured for serially adding eleven 8-bit numbers inside a single-cycle timed loop, and the FPGA performed a maximum of 104123 16-bit 1024-point FFTs per second.[23] These benchmarking data are for the Virtex-5 LX50, while the RIO Evaluation Kit contains a Spartan-6 LX45 which was not included in the tests but is better because the logic cells have a dual-register 6-input lookup table structure with an additional flip-flop. Designs require fewer logic levels, leading to less delay and up to 25% more system throughput.[24] The Spartan-6 has 6822 slices, where a slice is 4 LUTs and 8 FFs. Meanwhile, the Virtex-5 LX50 has 7200 slices where a slice is 4 LUTs and 4 FFs.[25]

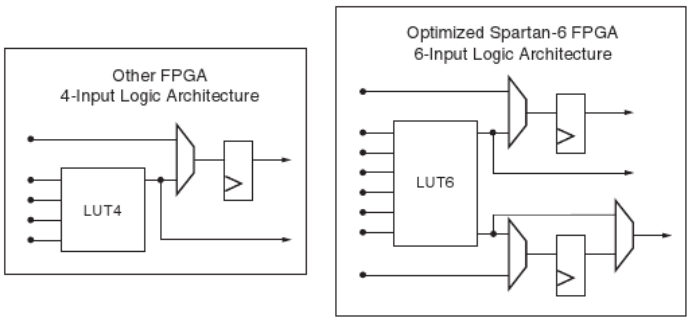


Figure 16: Spartan-6 architecture.[24]

Chapter 3: Discussion

3D image processing for OCT data

These results show that LabVIEW and ImageJ can be used to create 3D OCT images as well as apply various filters to them. The benefit of this approach is its ability to handle large datasets, because the binary file format conserves space and the amount of memory allocated to ImageJ may be increased through Edit>Options>Memory and Threads. Also, both these programs run on multiple operating systems. To make the images look better, a color lookup table could be added.

FPGA implementation of FFT

Spectral (Fourier) domain OCT depends on various signal processing techniques in order to render images. Each of these mathematical steps, outlined in Fig. 17, would benefit from faster computation and parallelism possible through use of an FPGA.

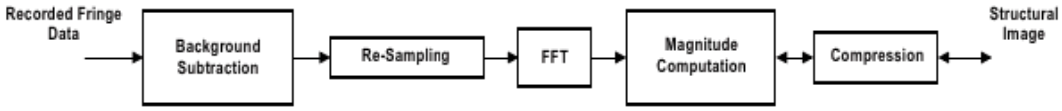


Figure 17: General model of signal processing in OCT.[9]

Perhaps the most important of these steps is the fast Fourier transform (FFT), whose FPGA implementation is demonstrated in the previous section. FFT reconstructs the axial scan as a function of depth, since the interference fringes were measured in the spectral domain in terms of wavenumber (k).[9] Li (2011)[20] explains a way to implement OCT algorithms on a Virtex-5 LX155T, using 2-byte fixed point representation. Each step is connected by two 4-byte Fast Simplex Links (FSL), 2 bytes real and 2 bytes imaginary. FSL is a Xilinx IP core for unidirectional FIFO. For background subtraction, the Xilinx Subtractor IP core was used to remove the DC component due to the reference

arm. High clock rates were made possible by the high speed DSP slices on the FPGA. The DC level is acquired before processing, assumed constant, and stored on a pre-initialized ROM. Resampling uses two ROMs, also pre-initialized, one containing a lookup table for the new sample data order, and the other containing the linear interpolation coefficients. In addition, a buffer exists for temporary data storage. Two Xilinx 1024-point FFT cores implement the Hilbert transform for dispersion compensation. The complex multiplication is mapped onto DSP slices for optimal performance; the same should be done for butterfly arithmetic but was not, due to insufficient FPGA resources. Phase coefficients are pre-initialized in ROM as well, which are multiplied with the complex output of the Hilbert transform.

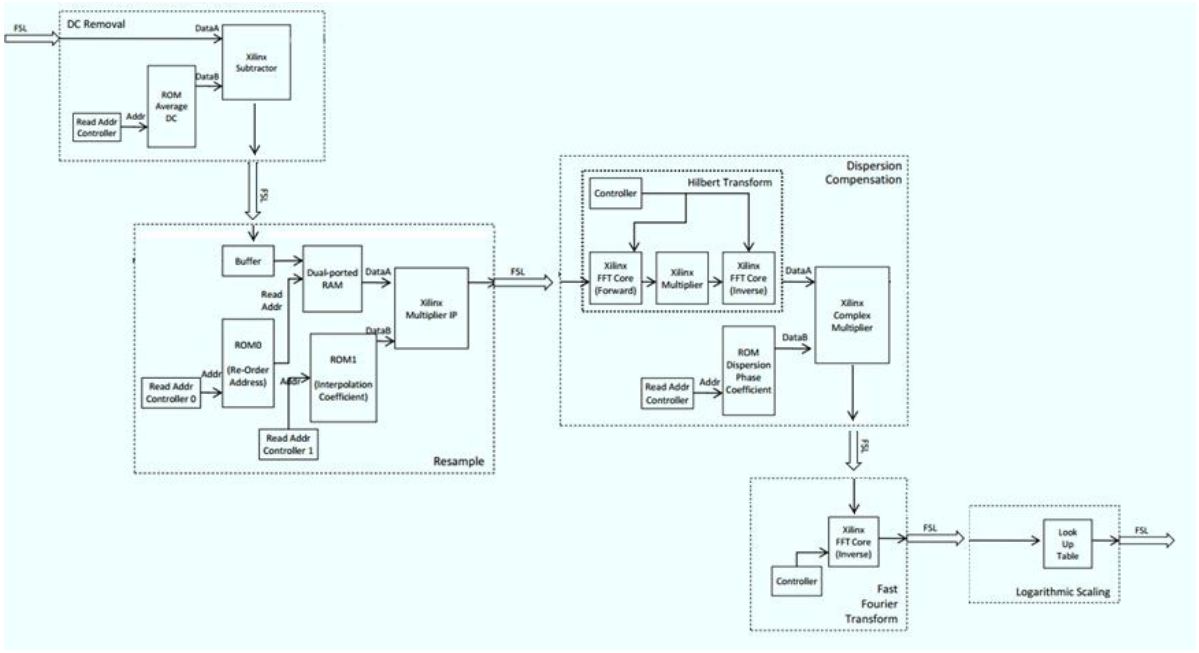


Figure 18: A specific implementation of OCT signal processing. [20]

For the FFT core, both the complex multiplier and butterfly arithmetic were mapped onto DSP slices, giving a 32-bit output which was truncated to 16 bits for

logarithmic scaling. Logarithmic scaling was also pre-initialized as a lookup table in ROM, with 64,000 entries each having 2 bytes address width and data width.[20]

FPGA-based processing would be beneficial to many methods of characterizing biological samples, not just OCT imaging. For example, photonic crystal microarray biosensors were previously researched and fabricated. The transmission spectrum of the photonic crystal waveguides shows minima corresponding to the resonant wavelength of each photonic crystal. Binding of analyte with probe causes a shift in the resonant wavelength, enabling label-free detection of biomolecules. Previous work has demonstrated that photonic crystals with submicron dimensions are highly sensitive to trace volumes of analytes.[10] While biosensing has been demonstrated in these devices with single biomolecules,[11-14] no previous effort has been made to extend device capability to microarrays. The device consists of an array of photonic crystal microcavity resonators coupled to a single photonic crystal waveguide that give rise to minima in the photonic crystal waveguide transmission spectrum at the resonance frequency of the microcavity, determined by binding events on the biomolecule coating. Photonic crystal microcavities will be coated with biomolecules that allow specific detection of bioagents and pathogens of interest, using a patterning technique with parylene microchannels[15] which keeps biomolecules separate, in solution and at room temperature or lower.

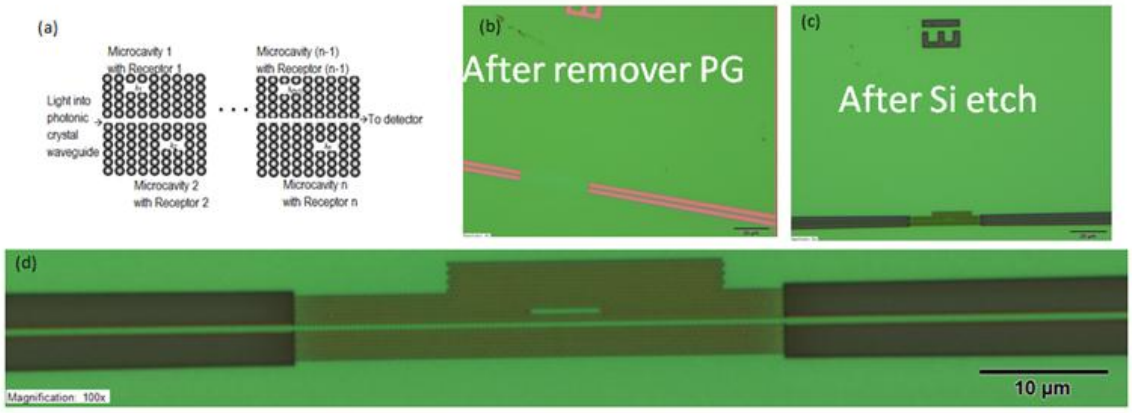


Figure 19: Photonic crystal (PC) biosensor. (a) Device principle. (b,c) Some fabrication steps. (d) Closeup of PC array.

In addition to facilitating OCT, FPGAs can be used to control multiple sensors. For example, Uzenkov et al. (2006)[16] monitored ten light sources generated by a fiber Bragg grating (FBG) strain sensor and separated by a monochromator. In their system, similarly to the photonic crystal sensors, center wavelength shift determines the magnitude of the stimulus applied. Light back-reflected from the grating array is imaged by a spectrometer onto an FPGA-controlled 2048-pixel linear CCD array at a rate of 2 MHz, equivalent to a new set of results every 35 ms.[16]

More recently, Zheng and Kang (2012)[17] have also constructed an FPGA-based FBG sensor analyzer. Their system also incorporates a DSP chip which is responsible for such tasks as filtering noise and detecting the signal peak. The FPGA acquires 17 channels of data and sends them to an SDRAM buffer of the DSP in a triangular wave cycle. An interrupt signals the DSP to start processing. Data are also sent to another DSP buffer during the next triangle wave period. When processing of each buffer is completed, the FPGA sends the results to its internal buffer, processes them, and sends them over Ethernet to a PC, which can control the programmable gain.[17]

Cao et al. (2011)[18] apply these concepts to embedded biosensors by developing FPGA algorithms for a porous silicon microcavity membrane. This device can generate different optical reflectance spectra for various molecular solutions. Shift of the resonant dip is detected to distinguish target molecule solutions of different concentrations. First, the noise (SNR=34.5 to 47.7 dB) is smoothed by filtering. The filtering is stronger with a larger window size, but too large of a window may filter out the resonant dip. They tried finite impulse response (FIR) low-pass filtering, average filtering, and median filtering, and found that for large noise, median filtering performs worse than either FIR or average filtering. In addition, the hardware implementation would be slow since the median filter mainly utilizes MUX and comparators for sorting. FIR filtering requires numerous multipliers and memories, while average filtering only requires a divider and some adders. The divider can be implemented simply by discarding the lower bits, if the data type is unsigned and the window size is a power of 2. Hence the researchers chose the average filter. Their algorithm for detecting the resonant dip proceeds as follows. The inputs are the data curve to be processed, and the dip size 2N+1, where N is set according to the SNR and the amount of data. The outputs are the coordinates of the target point. Slopes are calculated, where a positive slope is 1 and negative is 0. Then the point is found that has at least K points with negative slope to the left of it, and at least K points with positive slope to the right of it, where K is slightly smaller than N.[18]

Alternatively, one may wish to perform the processing on microcontrollers; to this end, Rao et al. (2012)[19] describe a FPGA-implemented addressable chip in which 16 sensors are mapped to three inputs of a microcontroller.

In conclusion, many possibilities exist for FPGA applications to bioimaging and biosensing. This thesis has discussed potential benefits of using an FPGA for processing

OCT data, such as increased processing speed and reduced instrument size. Other topics included in this thesis are Fast Fourier Transform implementation in LabVIEW FPGA, rendering and filtering of three-dimensional OCT images, and further applications such as FPGA-mediated integration of photonic crystal microarray biosensors.

Appendix: Windowing Filter Algorithm

```
% Length = numcols (value is 640 in this case)
% A(cluster)/B = numrows (value is 300 in this case)
% Both are constants (image dimensions)
% In labview this code is a for loop with a while and a case inside it. colindex is
the for loop
index, rowindex is feedback node.

rowindex=0;
colindex=0;

while((rowindex<(numrows-2))&&(colindex<(numcols-4))) {

     % run array subset vi here, with rowindex and colindex and lengths of 3 and 5

    if(colindex>=(numcols-4)) {

                    rowindex=(rowindex+3);
        colindex=0;}

    else{colindex++;};
  ;
 }; % end of while loop


if((rowindex>=(numrows-2))&&(colindex>=(numcols-4)))
{ % windowed image complete - would be empty case in LV but what about other
languages?};
% note, no edge cases for 3x5 window

Here is the labview version (on the repo):



if(((3*i)>=numrows)&&((5*i)>=numcols))
{ % windowed image complete - would be empty case in LV but what about other
languages?};
% note, no edge cases for 3x5 window
```

Say for example the image was only 6x10, then the code above is trying to do this, where the top left is (0,0):

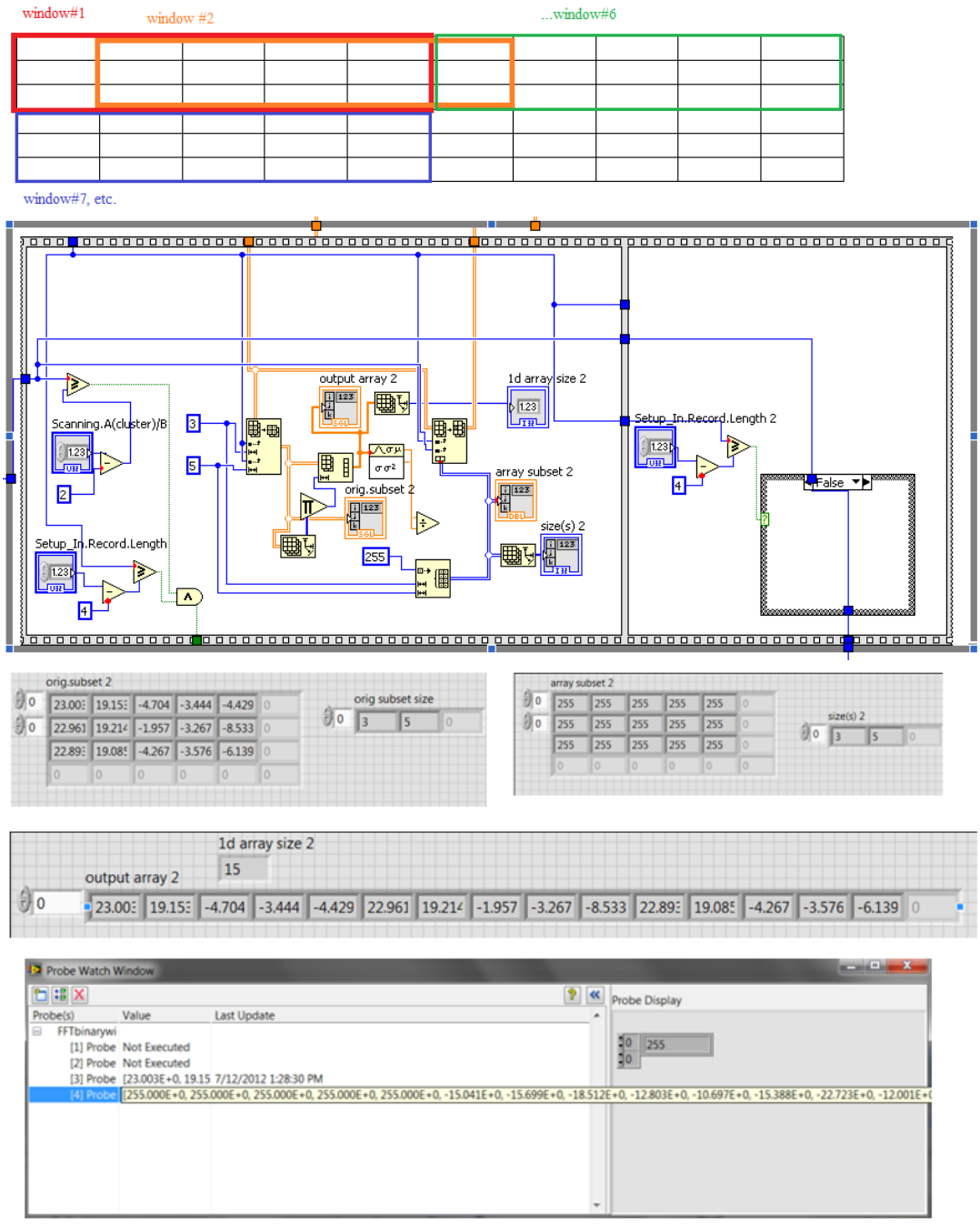


Figure 16: Windowing filter. (top) Diagram of what the code is supposed to do; (middle) code when translated into LabVIEW. (bottom) obtain an array subset, reshape it into 1D to do stats on it, and replace the original subset (for testing purposes use 255 instead of stats)

References

1. Suzuki, T. (2011). Using NI FlexRIO to Develop a High-Speed, Compact OCT Imaging System. Retrieved 10 Feb 2013 from http://sine.ni.com/cs/app/doc/p/id/cs-13335

2. Jung, W., Kim, J., Jeon, M., Chaney, E.J., Stewart, C.N., Boppart, S.A. (2011). Handheld Optical Coherence Tomography Scanner for Primary Care Diagnostics. IEEE Transactions on Biomedical Engineering, 58(3),741-744.

3. Ustun, T.E., Iftimia, N.V., Ferguson, R.D., Hammer, D.X. (2008). Real-time processing for Fourier domain optical coherence tomography using a field programmable gate array. Rev. Sci. Instrum. 79(11), 114301.

4. Nicolle, J.P. (2009). How FPGAs work. Retrieved 23 Feb 2013 from http://www.fpga4fun.com/FPGAinfo2.html

5. Li, J., Sarunic, M.V., Shannon, L. (2011). Scalable, High Performance Fourier Domain Optical Coherence Tomography: Why FPGAs and Not GPGPUs. 2011 IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp.49-56.

6. Xu, J., Ishikawa, H., Wollstein, G., Schuman, J.S. (2011). 3D optical coherence tomography super pixel with machine classifier analysis for glaucoma detection. 2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBC, pp.3395-3398.

7. National Instruments, Inc. (No date given.) NI RIO Evaluation Kit Tutorial.

8. Xilinx, Inc. (2011). LogiCORE IP Fast Fourier Transform v7.1 Product Specification, DS260.

9. Ali, M. & Parlapalli, R. (2010). Algorithms for Optical Coherence Tomography on TMS320C64x+. Texas Instruments Application Report SPRABB7.

10. Lee, M. & Fauchet, P. (2007). Two-dimensional Si photonic crystal microcavity for single particle detection, Proceedings of the 4th IEEE International Conference Group IV Photonics pp. 1-3.

11. Topolancik, J. et al. (2003) Fluid detection with photonic crystal based multichannel waveguides, Appl. Phys. Lett. 82, 1143-1145 (2003).

12. Buswell, S.C., et al. (2008) Specific detection of proteins using photonic crystal waveguides, Optics Express 16 (20), 15949.

13. Skivesen, N. et al. (2007). Photonic crystal waveguide biosensor, Optics Express 15 (6), 3169.

14. Chow, E. et al. (2004). Ultracompact biochemical sensor built with two-dimensional photonic crystal microcavity. Optics Letters 29 (10), 1093.

15. Atsuta, K. et al. (2007). A parylene lift-off process with microfluidic channels for selective protein patterning, Journal of Micromechanics and Microengineering 17, 496.

16. Uzenkov, O., Lee, P., Webb, D. (2006). An FPGA based Measurement System for a Fibre Bragg Grating (FBG) Strain Sensor. Proc. IEEE Instrumentation and Measurement Technology Conference, pp.2364-2367.

17. Zheng, B.H. & Kang, Q.L. (2012). A Novel Fiber Bragg Grating Sensor Analyzer Based on FPGA and DSP Platform. Computational Intelligence and Design (ISCID), 2012 Fifth International Symposium, vol.2, pp.311-314.

18. Cao, Y.J., Zhu, Y.X., Rong, G.G., Cheng, G., & Qiu, M.K. (2011). Efficient Pattern Detection for Embedded Optical Bio-sensing System. Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference, pp.259-266.

19. Rao, M.V.G., Kumar, P.R., Prasad, A.M. (2012). Design and FPGA implementation of Addressable Chip for Multiple Sensor Applications. International Journal of Technological Exploration and Learning, 1(1).

20. Li, J. (2011). Accelerating Fourier Domain Optical Coherence Tomography using general purpose graphics processing units and field programmable gate arrays (Master's thesis). Retrieved from https://theses.lib.sfu.ca/sites/all/files/public_copies/etd6450_jli_pdf_63801.pdf

21. National Instruments, Inc. (2011). Transferring Data between Devices or Structures Using FIFOs. Retrieved 6 Apr 2013 from http://zone.ni.com/reference/en-XX/help/371599G-01/lvfpgaconcepts/fpga_transfer_data/

22. Milewski, B. (2006). Fast Fourier Transform (FFT). Retrieved 6 Apr 2013 from http://relisoft.com/Science/Physics/fft.html

23. National Instruments, Inc. LabVIEW FPGA Benchmarks for Virtex-5 R Series Targets. Retrieved 6 Apr 2013 from http://www.ni.com/white-paper/7242/en

24. Smerdon, M. (2011). High-Volume Spartan-6 FPGAs: Performance and Power Leadership by Design. Retrieved 6 Apr 2013 from www.xilinx.com/support/documentation/white_papers/wp396_S6_HV_Perf_Power.pdf

25. National Instruments, Inc. (2012). How Many Slices Does My FPGA Chip Have? Retrieved 6 Apr 2013 from http://digital.ni.com/public.nsf/allkb/A7613AD731C13BCB86257995007F5D4F

26. National Instruments, Inc. (2009). The Fundamentals of FFT-Based Signal Analysis and Measurement in LabVIEW and LabWindows/CVI. Retrieved 29 April 2013 from www.ni.com/white-paper/4278/en